NATIONAL AERONAUTICS
AND SPACE ADMINISTRATION

Bottom-up Software Reuse: Documentation

James Marshall

Innovim / NASA GSFC

5th ESDS WG Meeting

Nov. 14, 2006

About Documentation

- Documentation provides readers with the additional information they need about the software or system.
- Different types of documentation:
  - *User* documentation for end users to use the software/system
  - *Process* documentation for managers to use in planning, budgeting, and scheduling software processes
  - *System* documentation on structure, components, and interactions for developers
- Examples of documentation standards:
  - NASA-STD-2100-91, NASA Software Documentation Standard
  - IEEE Std 1063-2001, Standard for User Documentation
  - ISO/IEC 18019:2004, Guidelines for the Design and Preparation of User Documentation for Application Software
- In addition, reusable software components should have a corresponding reuse manual.

Motivation for a Reuse Manual

- A reuse manual provides information that enables the evaluation, understanding, use, and adaptation of a software component.
- Answers questions such as:
  - What kind of component is it?
  - What is the component's functionality?
  - Can the component be reused in our context? How?
  - What else is needed to reuse the component?
  - Can the component be customized/adapted/modified? How? To what extent?
  - Can the component be interconnected with our components?
  - Is the component's quality sufficient for our purposes?
- General structure of information may be: general, reuse, administrative, evaluation, other.

Reuse Manual, Suggested Form

- General information
  - Introduction
  - Classification
  - Functionality
  - Platforms
  - Reuse status
- Reuse information
  - Installation
  - Interface descriptions
  - Integration and reuse
  - Adaptation
- Administrative information
  - Procurement and support
  - Commercial and legal restrictions
  - History and versions
- Evaluation information
  - Specification
  - Quality
  - Performance and resource requirements
  - Alternative components
  - Known bugs/problems
  - Limitations and restrictions
  - Possible enhancements
  - Test support
  - Interdependencies
- Other information
  - System documentation
  - References
  - Reading aids

Self-documenting Code

- Source code can serve as internal documentation.
- Good programming style is the key.
  - Good program and logical structure
  - Straightforward, easy to understand approaches
  - Good choice of names for variables, routines, classes, etc.
  - Use of named constants instead of literals
  - Clear layout with good formatting
  - Minimized control flow and data structure complexity
- Benefits include simplicity, clarity, completeness, consistency, and robustness.
- Self-documenting code does not rely on comments, but they can be helpful.

Some Examples

Bad Style = Poor Documentation

```
for ( i = 2; i <= num; i++ ) {
meetsCriteria[ i ] = true;
}
for ( i = 2; i <= num / 2; i++ ) {
j = i + i; while ( j <= num ) {
meetsCriteria[ j ] = false;
j = j + i;
}
}
for ( i = 1; i <= num; i++ ) {
if ( meetsCriteria[ i ] ) {
System.out.println ( i + " meets
    criteria." );
}
}
```

Both Java code fragments do the same thing, but one uses descriptive variable names and a clear layout, making it more readable and self-documenting.

Good Style = Good Documentation

```
for ( primeCandidate = 2; primeCandidate <=
    num; primeCandidate++ ) {
    isPrime[ primeCandidate ] = true;
}

for ( int factor = 2; factor < ( num / 2 );
    factor++ ) {
    int factorableNumber = factor + factor;
    while ( factorableNumber <= num ) {
        isPrime[ factorableNumber ] = false;
        factorableNumber = factorableNumber +
    factor;
    }
}

for ( primeCandidate = 1; primeCandidate <=
    num; primeCandidate++ ) {
    if ( isPrime[ primeCandidate ] ) {
        System.out.println( primeCandidate + "
    is prime." );
  }
}
```

Note that there are no comments in either code fragment, but this one is still more readable.

Commenting Source Code

- Comments can have a negative effect if they:
  - Repeat the code
  - Provide unnecessary explanations
    - Try to improve complex/tricky code rather than commenting it
  - Were entered as temporary markers
    - Fix problems and remove markers

```
// Increments the counter i
i = i + 1;
employees[i].processPayments();
```

- Comments are beneficial when they:
  - Summarize the code
  - Describe the code's intent
    - e.g., "get current employee info" rather than "update employeeRecord object"
  - Provide information the code can't express itself
    - Copyrights, confidentiality notices, version numbers, etc.

```
// Scan through all employees
   in database
i = i + 1;
employees[i].processPayments();
```

Tools such as Javadoc can also use comments to generate documentation.

References

- Software Engineering with Reusable Components by Johannes Sametinger
  - Types of documentation and reuse manual information
- Code Complete, Second Edition by Steve McConnell
  - Self-documenting code information and examples
- General Coding Best Practices course on https://nasa.skillport.com/
  - Commenting source code, benefits of self-documenting code